**Human Body Skeleton Detection and Tracking using OpenCV**

*Computer Vision Project Report*

**Problem Statement**

Pose Estimation is a general problem in Computer Vision where we detect the position and orientation of an object using OpenCV. This usually means detecting key point locations that describe the object. The proposed system uses video stream input through an integrated webcam and processes it to obtain the basic joints of the human skeleton. The input video stream is processed and sampled frame by frame. All the major body joints are identified and tracked by our proposed model. Skeleton model is developed using the obtained joints. Obtained skeleton is properly tracked to obtain real time results. The system is developed with final aim of taking it on handheld devices like Smartphone, tablets etc.

**Introduction**

Human pose estimation is typically formulated probabilistically to account for ambiguities that may exist in the inference (though there are notable exceptions). In such cases, one is interested in estimating the posterior distribution, p(x|z), where x is the pose of the body and and z is a feature set derived from the image. The key modeling choices that affect the inference are: – The representation of the pose   
– The nature and encoding of image features   
– The inference framework required to estimate the posterior  
– p(x|z)

Next, the primary lines of research in pose estimation with respect to these modeling choices are reviewed. It is worth noting that these three modeling choices are not always independent. For example, some inference frameworks are specifically designed to utilize a given representation of the pose.

Traditional Methods
Figure Drawing
â Cylinders for
each body parts
â Join up the
cylinders
Pictorial Structures
â Unary te...

Fig: Tradional approaches to human pose estimation

The problem of human pose estimation, defined as the problem of localization of human joints, has enjoyed substantial attention in the computer vision community. In Fig. 2, one can see some of the challenges of this problem – strong articulations, small and barely visible joints, occlusions and the need to capture the context. The main stream of work in this field has been motivated mainly by the first challenge, the need to search in the large space of all possible articulated poses. Part-based models lend themselves naturally to model articulations and in the recent years a variety of models with efficient inference have been proposed. The above efficiency, however, is achieved at the cost of limited expressiveness – the use of local detectors, which reason in many cases about a single part, and most importantly by modeling only a small subset of all interactions between body parts. These limitations have been recognized and methods reasoning about pose in a holistic manner have been proposed but with limited success in real-world problems. In this work we ascribe to this holistic view of human pose estimation.

Challenges
2015/9/11 5
 

Fig: Challenges to human pose estimation

**Failure Cases**

• Articulation

• Fore-Shortening

• Occlusions and distractions

• Cluttered background or overlapping people

Failure Cases
â¢ articulation
â¢ fore-shortening
â¢ occlusions and distractions
â¢ cluttered background or overlapping people
...

Fig: Incorrect articulation of different poses

**What is articulated body pose estimation?**

Recovers the pose of an articulated body, which consists of joints and rigid parts using image-based observations.

Results on LSP dataset
2016/8/11 17
 

Fig: Skeleton Outline after articulation

**Motivation**

* Motion Sensing technology is increasingly deployed in many applications where human detection is required such as gaming, security and military.
* Motion sensing input device such as Microsoft’s Xbox 360 Kinect provides this applicability using infrared sensors. **IR sensors** are much more **expensive** compared to optical cameras.
* Installation procedure is hectic and inconvenient to be widely used.
* Pose Estimation is a general problem in Computer Vision where we detect the position and orientation of an object. This usually means detecting key point locations that describe the object.

**Objective**

A model is developed to detect the major key points of the body in the image/video using the image/video as the input stream and track the movements of the key – points using OpenCV to reduce the overheads of installation of Motion Sensing Technology which uses Infrared Sensors

The goal of this model is to advance the state-of-the-art of articulated pose estimation in scenes with multiple people. To that end we contribute on three fronts. We propose

(1) improved body part detectors that generate effective bottom-up proposals for body parts  
(2) novel image-conditioned pairwise terms that allow to assemble the proposals into a variable number of consistent body part configurations  
(3) an incremental optimization strategy that explores the search space more efficiently thus leading both to better performance and significant speed-up factors.

Evaluation is done on two single-person and two multi-person pose estimation benchmarks. The proposed approach significantly outperforms best known multi-person pose estimation results while demonstrating competitive performance on the task of single person pose estimation

**Implementation**

**Datasets used are:**

* COCO dataset
* MPII dataset

**Model Architecture**

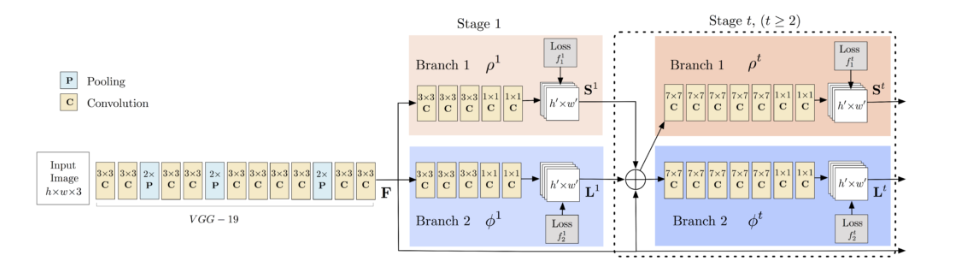


Fig: **Multi-Person Pose Estimation model architecture:** The model takes as input a colour image of size w × h and produces, as output, the 2D locations of key points for each person in the image.

**Implementation**

**Stages:  
Stage 0**: The first 10 layers of the VGGNet are used to create feature maps for the input image.

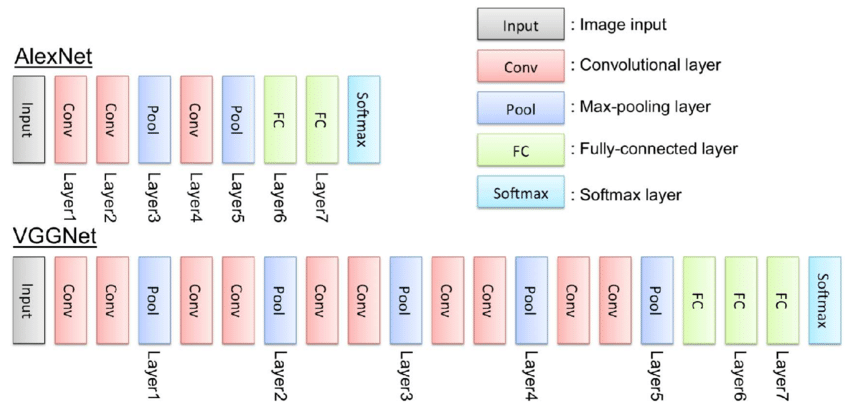
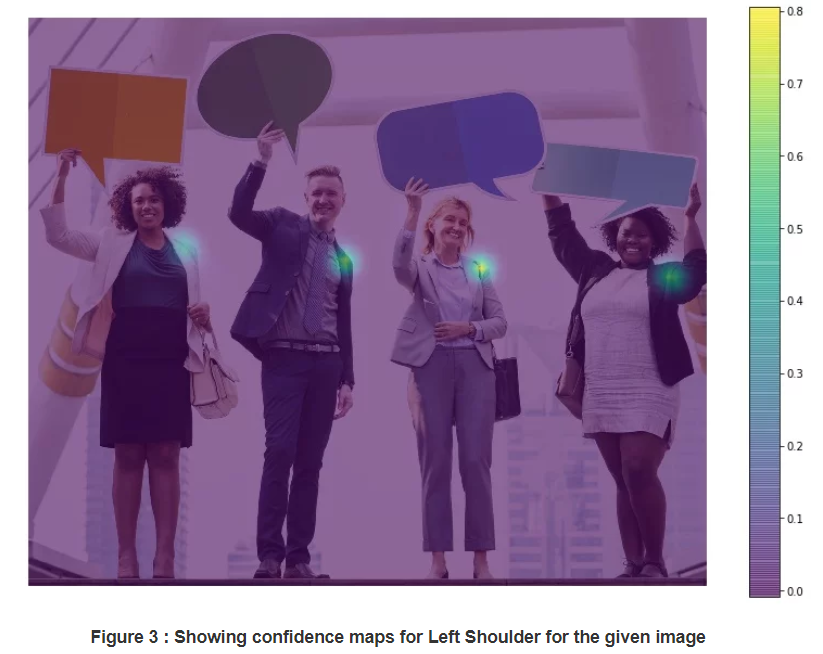


Fig: VGGNet Architecture

* **Stage 1**:   
  Branch 1 output – predicts confidence maps of different points of the body.

Branch 2 output - a set of 2D vector fields of **part affinities**, which encode the **degree of association** between parts.

Fig: Showing Confidence map for left shoulder

* **Stage 2:** Produces all the key points of the body in the image.

**ConvNet Architecture:**

A simple ConvNet for classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

* INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
* CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.



Fig: A regular 3-layer Neural Network.



Fig: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

* RELU layer will apply an elementwise activation function, such as the max(0,x)max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
* POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
* FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.



Fig: The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full web-based demo is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don’t. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

The whole VGGNet is composed of CONV layers that perform 3x3 convolutions with stride 1 and pad 1, and of POOL layers that perform 2x2 max pooling with stride 2 (and no padding). We can write out the size of the representation at each step of the processing and keep track of both the representation size and the total number of weights:

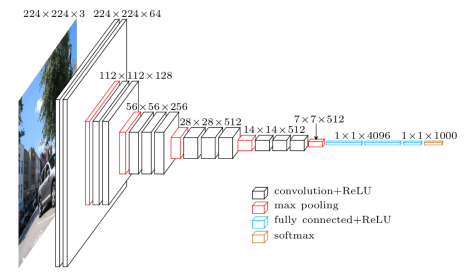


Fig: VGGNet Working

As is common with Convolutional Networks, notice that most of the memory (and also compute time) is used in the early CONV layers, and that most of the parameters are in the last FC layers. In this particular case, the first FC layer contains 100M weights, out of a total of 140M.

Computational Considerations

The largest bottleneck to be aware of when constructing ConvNet architectures is the memory bottleneck. Many modern GPUs have a limit of 3/4/6GB memory, with the best GPUs having about 12GB of memory. There are three major sources of memory to keep track of:

* From the intermediate volume sizes: These are the raw number of **activations** at every layer of the ConvNet, and also their gradients (of equal size). Usually, most of the activations are on the earlier layers of a ConvNet (i.e. first Conv Layers). These are kept around because they are needed for backpropagation, but a clever implementation that runs a ConvNet only at test time could in principle reduce this by a huge amount, by only storing the current activations at any layer and discarding the previous activations on layers below.
* From the parameter sizes: These are the numbers that hold the network **parameters**, their gradients during backpropagation, and commonly also a step cache if the optimization is using momentum, Adagrad, or RMSProp. Therefore, the memory to store the parameter vector alone must usually be multiplied by a factor of at least 3 or so.
* Every ConvNet implementation has to maintain **miscellaneous** memory, such as the image data batches, perhaps their augmented versions, etc.

Once you have a rough estimate of the total number of values (for activations, gradients, and misc), the number should be converted to size in GB. Take the number of values, multiply by 4 to get the raw number of bytes (since every floating point is 4 bytes, or maybe by 8 for double precision), and then divide by 1024 multiple times to get the amount of memory in KB, MB, and finally GB. If your network doesn’t fit, a common heuristic to “make it fit” is to decrease the batch size, since most of the memory is usually consumed by the activations.

**Applications**

* Can be used by the **sport authorities** to **live track the player’s movements** and detect the key points of the body to understand the injury reasons if occurred any.
* Can be used for clothes parsing.
* Can be used to live-track human body movements and analyse them.
* Can be used to understand the **gameplay of a player** and his next moves to **improve the game strategy**.
* **Orthopaedic patient diagnosis** - can be used to by the Doctors to understand the key points while analysing the body postures of the patients.
* Can be used by the **gymnastic person** to improve their movements in order to prevent any **future injuries**.
* **Surveillance Activities** - Action and behaviour analysis, detection of abnormal activities.

Applications
Action recognition Clothing Parsing
Gaming
2015/9/11 4
Human tracking
 

Fig: Some of the applications of human pose estimation

**Results**

Convolutional pose machines provide an end-to-end architecture for tackling structured prediction problems in computer vision without the need for graphical-model style inference. We showed that a sequential architecture composed of convolutional networks is capable of implicitly learning a spatial models for pose by communicating increasingly refined uncertainty-preserving beliefs between stages. Problems with spatial dependencies between variables arise in multiple domains of computer vision such as semantic image labeling, single image depth prediction and object detection and future work will involve extending our architecture to these problems. Our approach achieves state of the art accuracy on all primary benchmarks, however we do observe failure cases mainly when multiple people are in close proximity. Handling multiple people in a single end-to-end architecture is also a challenging problem and an interesting avenue for future work.

 **Input Image**

****

**Input Image**



**Output-Key Points**

****

**Output-Skeleton**

**Future Scope**

We can work on pose detection for multiple people using VGGNet to train our model to detect skeleton points for multiple people in a scenario with multiple people are present to detect them and thus increase the efficiency of our model.